Fragile Watermarking of 3D Models in a Transformed Domain

applied sciences

MDPI

Article

Fragile Watermarking of 3D Models in a Transformed Domain

Marco Botta [1,*], Davide Cavagnino [1], Marco Gribaudo [2] and Pietro Piazzolla[3]

[1] Dipartimento di Informatica, Università di Torino, 10100 Torino, Italy; davide.cavagnino@unito.it
[2] Dipartimento di Elettronica, Informazione e Bioingegneria—Politecnico di Milano, 20100 Milano, Italy; marco.gribaudo@polimi.it
[3] Dipartimento di Ingegneria Gestionale e della Produzione—DIGEP—Politecnico di Torino, 10100 Torino, Italy; pietro.piazzolla@plm.polito.it
* Correspondence: marco.botta@unito.it

Received: 7 April 2020; Accepted: 2 May 2020; Published: 7 May 2020

Abstract: This paper presents an algorithm aimed at the integrity protection of 3D models represented as a set of vertices and polygons. The proposed method defines a procedure to perform a fragile watermarking of the vertices' data, namely 3D coordinates and polygons, introducing a very small error in the vertices' coordinates. The watermark bit string is embedded into a secret vector space defined by the Karhunen–Loève transform derived from a key image. Experimental results show the good performance of the method and its security.

Keywords: 3D model; integrity protection; fragile watermarking; data hiding; genetic algorithm; Karhunen–Loève transform

1. Introduction

Digital watermarking refers to a family of methods that embed a signal into a digital object. For example, visible logos in a TV broadcast or on a picture for sale on the web are two applications of visible watermarking. Nonetheless, digital technology allows for more complex applications of the general idea of watermarking, also allowing invisible (to humans) watermarks to be embedded into digital objects. Copyright protection, tracking origin, authentication, and integrity protection are examples of the application of this technology.

It is possible to classify watermarking algorithms according to various characteristics [1]: the main properties of digital watermarking algorithms will be discussed in the following.

A first, trivial, subjective property of watermarks is imperceptibility, that is the inability of a human being to recognize the presence of a watermark embedded into an object.

A digital watermark may be robust, semi-fragile, or fragile. Robust watermarking refers to the embedding of a signal with the purpose of resisting malicious attempts aimed at its removal, maintaining the possibility to show the presence of the signal; a typical application of this kind of watermarking algorithms is copyright protection. On the other hand, fragile watermarks have the purpose of being altered by minimal modification of the digital object containing them; fields of application are integrity protection and authentication. The class of semi-fragile watermarking algorithms contains those methods that accept a minimal level of modification without flagging a tampering alarm.

On the detection side, watermarking methods may be grouped as informed (non-blind) or blind depending on the necessity or not of the original host object. Moreover, if an algorithm can recover the original data, it is called reversible; otherwise, it is called lossy or non-reversible.

The values of the watermark signal w may be embedded into the host object by modifying the object features in different domains; the most widely used are the spatial domain and the frequency

domain (or transformed domain). The spatial domain refers to the values that describe and represent the object; as examples, for an image, they may be the pixel intensities; for a 3D model, they may be the vertices' (spatial) coordinates. Instead, in the case of embedding in the frequency domain, the object's data are first transformed with a linear transform, like the discrete Fourier transform, the wavelet transform, the singular value decomposition or the Karhunen–Loève Transform (KLT), then the watermark signal is applied to alter the transform coefficients; after that, the inverse transformation is applied to compute the modified values, which represent the watermarked object. In general, embedding in the frequency domain requires satisfying non-linear constraints (like obtaining integer pixel values or modifying only some parts of the transformed coefficient); thus, computational intelligence algorithms which look for almost optimal solutions in non-linear domains can be a possible embedding method.

Most of the 3D model watermarking works are focused on copyright protection (by inserting robust watermarks), while there are fewer works on integrity protection and authentication. Moreover, existing approaches to fragile watermarking introduce high distortion and/or might require extra information for verification. In this paper, we present a blind, non-reversible fragile watermarking algorithm that adds an extra level of security (by defining a secret embedding space through the KLT transform) and protects the integrity of 3D models, with negligible alterations to the host model vertices' coordinates.

The host model original file format may be both textual or binary, but the watermarked model file is binary (as in many graphics file formats, like Blender).

The following section recalls some related works in the field of 3D model watermarking. Sections 3 and 4 present two fundamental tools used by the proposed algorithm, discussed in Section 5, to embed in a secure way a watermark in a 3D model. Section 6 reports numerical results from experiments applying the method to publicly available models. The results are discussed in Section 7, and some conclusions are drawn in Section 8.

2. Related Works

The field of digital watermarking has seen a large development of methods for images and audio, but also the protection of 3D models has received attention from researchers. Several algorithms for robust (see Table 1) and fragile (see Table 2) watermarking of 3D models have been developed in recent years. Here, we report a list of the properties declared by the authors, along with their pros and cons.

Table 1. Summary of related works on robust watermarking: main properties, pros, and cons.

Reference	Embedding Space	Verification Type	Pros	Cons
[2]	Spread spectrum	Non-blind	Watermark process as matrix operation	Requires original for detection, low quality
[3]	Spread spectrum	Blind	Redundant watermark embedding in sub-meshes for improved robustness	Complexity in computing eigenvectors
[4]	Spread spectrum	Blind	More efficient eigenvalue decomposition, more robustness	Low quality
[5]	Coarse, low resolution approximation	Blind	Pretty robust to attacks	Tampering detection relies on a simple correlation
[6]	Vertices' space	Non-blind	Redundant watermark embedding for robustness	Requires original for detection, low quality
[7]	Spherical harmonic transform space	Blind	Embeds in the transform space, robust	Low quality
[8]	Vertex norms distribution histogram bins	Blind	Robust against distortionless and distortion attacks	Not applicable to very small models and vulnerable to center of gravity alteration
[9]	Sparse quantization index modulation space	Blind	Uses a neural network to choose embedding vertices	Only robust to deletion of vertices

Table 2. Summary of related works on fragile watermarking: main properties, pros, and cons. KLT, Karhunen–Loève Transform.

Reference	Embedding Space	Verification Type	Pros	Cons
[10]	Vertices' space	Blind, semi-fragile	Resists unintentional changes (like compression, transformation, and random noise)	False positives possible
[11]	Vertices' space	Blind	Protection against cropping	Need inserted watermark for verification
[12]	Vertices' space	Blind	Addresses causality and convergence problems, users can control distortion level	Low quality
[13]	Mantissa of vertices' coordinates	Blind	Numerically stable, solves causality problem, immune to vertex renumbering, can control watermark intensity	Sensitive to parameter selection
[14]	Model connection points	Blind, semi-fragile	Watermark depends on model topological features	Can only authenticate and verify the topology integrity
[15]	Spherical coordinate space	Non-blind	Immune to vertex reordering, causality, convergence, and synchronization problems	Requires the center of gravity for detection
[16]	Hash transform space	Blind	Capable of detecting object cropping	Fails to localize changes and proved weak against vertex reordering
[17]	Vertices' space	Blind	Immune to vertex reordering	Low quality
[18]	Watermark digest	Blind	Computes a message digest of the model	Watermark requires extra space
[19]	Vertices' space	Blind	Robust to translation, rotation, and uniform scaling, but is fragile and sensitive to other operations	Low quality
[20]	Vertices' space	Non-blind	Robust to translation, rotation, and uniform scaling	Medium quality
Proposed Method	Secret KLT transform space	Blind	Imperceptible, highly secure, very fragile	False negatives

Our proposed method shares a number of features with the literature and differs substantially in others: the main and more significant difference is the embedding space, which is secret and allows an extra level of security, as it is unfeasible for an attacker to reproduce such a space. Most of the related works in the literature are actually robust watermarking schemes, while our method is an extremely fragile method whose watermark is broken by negligible alterations to the 3D models it protects. Similarly to [13], we modify the binary float representation of the vertices' coordinates, but our watermark is not directly inserted into them, as done in [13]. Moreover, the proposed method is immune to vertex reordering and can tolerate affine transformations to the model when these transformations are stored separately from the vertex coordinates in the output file format (such as done in Blender). Finally, similarly to [20], we use a genetic algorithm to compute a (quasi)optimal solution to the embedding problem, but again, our watermark is embedded into a secret space and not directly into the vertices' coordinates.

3. The Karhunen–Loève Transform

The discrete Karhunen–Loève Transform (KLT) (also called Hotelling transform or Principal Component Analysis (PCA)) is a linear transformation that maps vectors from one n-dimensional vector space to another vector space of the same dimension: this mapping is defined by a square matrix of size $n \times n$, called the kernel.

Differently from other linear transformations like the discrete cosine transform or the Fourier transform, one of the characteristics of the KLT is having a kernel that is not fixed a priori, i.e., its transformation matrix is computed from a set of vectors that must be given to define the mapping.

This characteristic of the KLT is exploited in the present algorithm to have a compact, secure, and efficient way to define a secret space of embedding: if the set of vectors used in computing the KLT kernel is kept secret, then also the kernel will be secret, defining a transformed domain in which to embed the watermark securely; for example, using a secret image's pixel values to define these vectors, then only the entities possessing this image will be able to verify the integrity of the 3D model. Moreover, lacking knowledge of the embedding space does not allow modifying the 3D model without altering the fragile watermark.

Consider a set of vectors $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_r\}$ of an n-dimensional vector space, and compute their mean vector $\mathbf{m} = E\{\mathbf{x}_i\}$. Then, evaluate the covariance matrix of the centered vectors $\mathbf{C} = E\{(\mathbf{x}_i - \mathbf{m})\,(\mathbf{x}_i - \mathbf{m})'\}$, where $\mathbf{z}'$ represents the conjugate transpose of $\mathbf{z}$. The eigenvectors $\mathbf{e}_i, i = 1, 2, \ldots, n$, of the covariance matrix $\mathbf{C}$, computed as:

$$\mathbf{C}\mathbf{e}_i = \lambda_i \mathbf{e}_i \tag{1}$$

define an orthonormal basis for the n-dimensional vector space; the λ_is are the associated eigenvalues. Thus, arranging the eigenvectors as rows of a matrix $\mathbf{A}$ in descending sort order of the respective eigenvalues, the KLT $\mathbf{y}$ of a vector $\mathbf{x}$ is computed as:

$$\mathbf{y} = \mathbf{A}(\mathbf{x} - \mathbf{m}). \tag{2}$$

Each element of the vector $\mathbf{y}$ is called the coefficient of the transform; the position of every coefficient in the vector $\mathbf{y}$ is called the order of the coefficient.

It is possible to reverse the transformation process through the inverse KLT:

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{y} + \mathbf{m} \tag{3}$$

which, by the orthonormality of $\mathbf{A}$, may be also written as:

$$\mathbf{x} = \mathbf{A}'\mathbf{y} + \mathbf{m}. \tag{4}$$

One property of the eigenvectors is that they define the directions of maximum data spread in the data sample, and sorting them according to their associated eigenvalues allows exploiting the energy compaction property of the KLT transform, useful when expressing the elements of the vector space with a reduced number of coefficients: it may be shown that the eigenvalues represent the variance of the coefficients in each dimension.

For more details on the KLT, see [21].

4. Genetic Algorithms

A Genetic Algorithm (GA) is a computing paradigm that simulates, in a simplified manner, the evolution and selection processes of natural species.

It may be used in solving non-linear optimizations problems when they may be coded in a set of parameters, and it is possible to define a function, called the fitness (function), that measures how close a solution is to the optimum.

A set of instantiated parameters describing the problem at hand is called the individuals: these parameters are put in a sequence similar to a chromosome, which is distinctive for its individual and defines it. Moreover, a fitness function is defined to evaluate the quality of the individual in terms of approximating the optimal solution.

A set of individuals, i.e., a population, is evolved in a way similar to the development of natural species. They mate, reproduce, and possibly have random modifications of their genes.

A chromosome codes a sequence of parameters, typically binary or integer numbers or a mix of them (but any type of data could be used in principle) with the constraint that the operations on them keep the data types consistent in the resulting chromosomes.

A population of individuals is created and (randomly) initialized; in general, a population of a hundred chromosomes is considered adequate.

Then, the GA goes through an iterative process, and in every cycle, the population is evolved according to the following steps:

- selection: pairs of individuals are mated for the following reproduction operation; individuals may be selected randomly or given priority depending on fitness; many strategies can be adopted, like roulette wheel or tournament selection [22];
- crossover: the previously selected pairs of chromosomes are mated by exchanging some (homologous) parts of their chromosomes with probability p_c; one or more "cutting" points are randomly chosen, and the genes between pairs of such points are exchanged to define two new offspring;
- mutation: with the aim of better exploring the solution space, each chromosome is mutated with probability p_m; mutation is performed by choosing at random one of the parameters coded in the chromosome and altering it by a random quantity;
- population update: to generate a new population and to keep the number of individuals constant, a selection among the individuals in the set composed by the old population and the new offspring should be performed; many different strategies could be adopted, e.g., two individuals in the set are randomly chosen, and the one with the best fitness is moved to the new population, repeating this process until the output population reaches the desired cardinality.

The described cycle is performed until a termination condition is met. In general, to avoid infinite cycles, an upper limit to the number of generations is fixed. Furthermore, the cycle may be terminated when at least one of the individual's fitness drops below (in this case, the lower the fitness, the better the individual) a pre-defined threshold. In both cases, the GA returns the best individual, representing the near-optimal solution it was able to find. For a deeper insight into GAs, see [22,23].

As will be shown in the following Section 5, to minimize the embedding noise due to watermark embedding, the watermark bits are stored, modifying the less significant part of the floating point numbers representing the 3D model vertex coordinates. This is obtained by considering the bytes encoding these floating point numbers as unsigned integers and changing the less significant bytes. This mode of operation results in a minimization problem in a non-linear space, a task where GAs, among other algorithms, are flexible and efficient to implement and apply.

5. The Proposed Algorithm

The developed algorithm has the objective to protect the shape and the structure of a 3D model, defined in terms of a mesh of polygons, providing a verification that allows for identification and localization of modifications to the model. In particular, the algorithm that will be presented in this section will protect:

- vertex coordinates (i.e., the geometry of the model);
- the polygon structure, as defined by the corresponding vertex connections (that is, the topology).

In the present embodiment, vertices' normal vectors are not considered because it was found that some 3D modeling software (e.g., Blender) alter them according to vertices' positions; thus,

authentication data would be obsoleted by the modeling software at saving or loading time, inducing our verification procedure to flag vertices as forged. Nonetheless, protecting vertices will have as a side effect the protection of normals computed from them.

To allow modifications that maintain the structure of the 3D model to the file representing it, the proposed method permits order alterations to the vertices and to the polygons as they are stored in the file. This provides independence, to some extent, from the particular file format.

The method is composed of two main modules, namely an embedder and an extractor, and three companion modules, which are a key generator, a watermark generator, and a verifier. The interactions between the modules and the main data interchanged are shown in Figure 1.

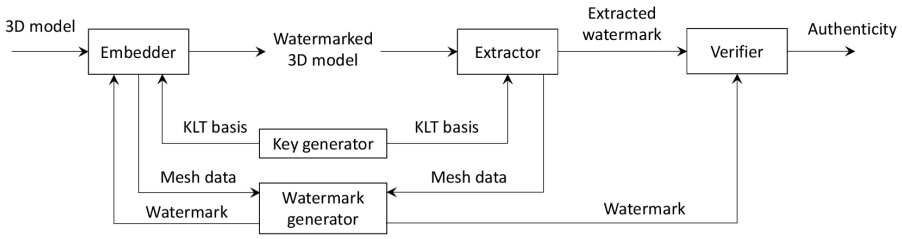


Figure 1. A high level scheme of the proposed method.

Each vertex is considered an Embedding Unit (EU); in the present embodiment, the vertex is defined by three spatial coordinates (x, y, z) and by the polygons to which the vertex belongs; a detailed description of this structure and of the embedding procedure will be given at the end of this section and in Section 5.2. The use of the EU concept solves the problem in 3D models that is not present in audio and image samples, that is the lack of the total order of the vertices.

A bird's eye-view of the various modules is the following:

- Key generator: defines a secret KLT basis to specify an embedding space;
- Watermark generator: creates the watermark bit string to be stored in each EU;
- Embedder: stores the watermark in the 3D model, one vertex at a time, using the KLT basis and the watermark provided by the respective modules; a computational intelligence technique, described in Section 4, is used to alter the vertex components so that the KLT coefficients of the vertex data contain the watermark; the output is a 3D model storing the fragile watermark;
- Extractor: extracts the watermark present in a 3D model using the secret KLT basis provided by the key generator module;
- Verifier: compares the extracted watermark with the one that should be contained in the vertices, providing the localization of tampered areas.

The input data are a 3D model composed of a set of vertices and a set of polygons. Each vertex is defined by three coordinates (x, y, z); every component value is considered in binary format, float representation (four bytes) in the IEEE 754 format. Every polygon is defined by the sequence of the indexes of its vertices. The watermark Embedding Unit (EU) is defined by the vertex data (x, y, z) along with a combined fingerprint of the t polygons to which the vertex belongs.

The fingerprint of a polygon is computed as follows: for every pair of vertices $(\mathbf{v}_i, \mathbf{v}_{i+1})$ encountered on the perimeter, a cryptographic hash (c. h.) function H is computed, $H(\mathbf{v}_i, \mathbf{v}_{i+1}) = h_{i,i+1}$; thus, for a polygon of n vertices, n cryptographic hashes will be obtained $h_{1,2}, h_{2,3}, \ldots, h_{n,1}$. To be independent from the starting vertex, these hashes are XORed $q = h_{1,2} \oplus h_{2,3} \oplus \ldots \oplus h_{n,1}$, then a c. h. is computed on the result q; the value $H(q)$ is the fingerprint of a polygon. Finally, a combined fingerprint F is obtained by XORing (to be independent from the order of the polygons considered) the fingerprints $H(q_i)$ of the t polygons the vertex belongs to:

$$F = H(q_1) \oplus H(q_2) \oplus \ldots \oplus H(q_t). \tag{5}$$

The c. h. function H used is MD5, whose hash length is 16 bytes.

Thus, the EU is composed of x, y, z (all floats, each one occupying four bytes) and a fingerprint F (bit string, 16 bytes), making 28 bytes in total (see Figure 2).

In Figure 2, note the bytes marked with vertical stripes: they are the least significant bytes of the mantissa of each float value. The watermark is embedded altering only those bytes; it follows that the polygon hashes do not take into account those bytes in the computation, i.e., if $[.]_3$ is the operator that extracts the three most significant bytes, then given two vertices $\mathbf{v}_i = (x_i, y_i, z_i)$ and $\mathbf{v}_j = (x_j, y_j, z_j)$, the c. h. $h_{i,j}$ is:

$$h_{i,j} = H([x_i]_3 \mid [y_i]_3 \mid [z_i]_3 \mid [x_j]_3 \mid [y_j]_3 \mid [z_j]_3), \tag{6}$$

where the symbol $\mid$ means string concatenation.

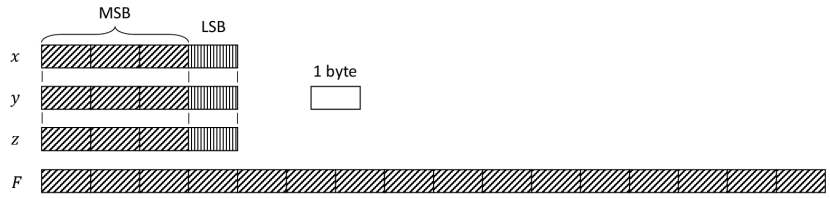


Figure 2. The fields and structure of an Embedding Unit (EU).

Note that the algorithm can cope with more data related to each vertex, for example *uv* coordinates: it is sufficient to add them to the EU and modify the KLT basis size.

5.1. Key Generation

The key generator module has the objective to create a secret orthonormal basis to define a secret space of embedding. This space must be known to the embedder and to the extractor; thus, the key generator must provide the basis to both of them during their operations.

In principle, any secret orthonormal basis of the required dimension (in the present embodiment, 28) would suffice; nonetheless, the KLT provides a method to define a basis starting from a set of vectors, so we found that using a secret image to derive a set of vectors from which to compute a KLT basis was a flexible and viable solution.

5.2. Watermark Embedding

The watermark string w to be embedded into an EU is derived by the key generator module from the fixed part of the EU itself, i.e., the diagonal striped bytes in Figure 2. This string can be computed as the MD5 hash of the aforementioned data: from the resulting 128 bits, a predefined subset is extracted for embedding. Note that this step is not strictly necessary because the watermark is embedded into a secret space, as will be discussed in the following; thus, also a constant w bit string would suffice; nonetheless, a variable w further improves security.

A high level description of the algorithm is the following.

The fragile watermark is stored in the coefficients of every EU KLT: the 28 bytes of the EU are considered as 28 non-negative integer pixel values, which are KLT transformed, producing 28 coefficients. If m is the payload in bpv (bits-per-vertex), i.e., m is the length of the watermark w for every EU, then a subset of m KLT coefficients is chosen a priori to store the watermark bits. In the present implementation, a coefficient c is considered as carrying a watermark bit b in position p (the p value is a parameter of the algorithm) iff:

$$b = \text{round}(2^{-p}c) \bmod 2 \tag{7}$$

(see [24] for a deeper discussion on various methods for embedding the watermark bits into a set of coefficients).

The GA alters the values of the vertical striped bytes in Figure 2 until the watermark is embedded into the EU KLT coefficients, i.e., if $\mathbf{g} = (g_1, g_2, \ldots, g_{28})$ is the EU, the GA alters g_4, g_8, g_{12} (the vertical striped bytes) so that the predefined coefficients in $(c_1, c_2, \ldots, c_{28}) = \mathrm{KLT}(g_1, g_2, \ldots, g_{28})$ contain the respective m bits watermark. The reason for using a GA in this task is its ability to find the solution of a non-linear problem (bytes representing integer values) looking for the (local) minima of the error w.r.t. the original data.

Referring to Equation (7), the present implementation of the algorithm considers the first m coefficients among the 28 transform coefficients and embeds into the bit in position $p = -2$.

When all the EUs have been processed by the GA, the 3D model with the new modified coordinates contains the fragile watermark allowing for the integrity check by the modules presented in the following section.

5.3. Watermark Extraction and Verification

The extraction of the watermark starts with building the EU for every vertex. Using the KLT basis from the key generator module, 28 coefficients are extracted from every EU: from the subset of coefficients used for watermark embedding, the watermark bits are computed using Equation (7) and stored in a string w_e. Note that this operation requires the computation of a forward KLT; the inverse KLT is not used in the proposed algorithm.

After that, the watermark generator module is called for every EU, producing a watermark string w for each of them.

At this point, the verifier module compares the two strings w_e and w for every vertex: if the strings are equal, then the vertex is marked as authentic; otherwise, it is flagged as potentially forged.

The possible forging of a 3D model represented as a set of vertices connected to form polygons may be done by altering or modifying:

- at least one of the coordinates of a vertex, or
- removing, replacing, or adding a vertex to a polygon, or
- shuffling the order of the vertices on the perimeter of a polygon.

According to the integrity protection provided by the proposed algorithm, a vertex can be detected as tampered if:

1. at least one of the coordinates x, y, z has been altered: in this case, if only the lower part (the vertical striped bytes in Figure 2) of one of the values is modified, then the vertex alone will be flagged as tampered; if the higher part is altered, then the vertex itself and all the vertices belonging to the polygons containing the vertex will be detected as tampered.
2. one of the polygons containing it is altered, that is:
    - one of its vertices has been modified in the higher part of the coordinates, or
    - at least one of its vertices has been substituted by another vertex, or
    - the order of the vertices on the perimeter has been changed (comprising the clockwise or anti-clockwise order), or
    - the number of vertices has been changed;
3. it has been removed from one of the polygons it belongs to, or
4. it has been added to a polygon (even duplicating a polygon).

6. Experimental Results

This section summarizes the results of a number of experiments we executed to test the performance of the proposed algorithm. In order to measure the resulting quality, we considered four objective parameters, namely the sensitivity, the Mean Absolute Error (MAE), the Root Mean Squared Error (RMSE), and the Peak Signal-to-Noise Ratio (PSNR). Indeed, the sensitivity gives a measure of the capability of the algorithm to detect tampering attacks. Thus, the meanings of the parameters used to measure the performance of the proposed algorithm are:

- Sensitivity (defined as in [24]): the sensitivity of level $\pm\delta$ is defined as the fraction of vertices that are detected as tampered in the verification phase when one byte of each EU is modified by $\pm\delta$; this is a very powerful fragility measure, in particular for small values of δ as one or two, because it is actually the least tampering an attacker can perform: any other attack, such as noise addition, adding, removing, or shifting vertices or surfaces, etc., would modify the mantissa values of the vertex coordinates of more than just $\pm\delta$;
- Mean Absolute Error (MAE): measures the average absolute difference between vertex attributes of the host and the watermarked model,

$$\mathrm{MAE} = \frac{\sum_{\mathbf{v}\in \mathrm{Y}} \sum_{c\in\Omega} |\mathbf{v}_c - \mathbf{v}_{c'}|}{\mathrm{card}(\mathrm{Y})\ \mathrm{card}(\Omega)}, \tag{8}$$

  where $\mathbf{v}$ scans the set Y of all vertices, c scans the set Ω of the attributes that can be modified by the embedder (i.e., in the present embodiment x, y, z), the $'$ symbol refers to the modified attribute in the watermarked model, and the function $\mathrm{card}()$ returns the cardinality of a set;
- Root Mean Squared Error (RMSE): measures the distortion between vertex coordinates of the host and the watermarked model,

$$\mathrm{RMSE} = \sqrt{\frac{\sum_{\mathbf{v}\in \mathrm{Y}} \|\mathbf{v} - \mathbf{v}'\|^2}{\mathrm{card}(\mathrm{Y})}}, \tag{9}$$

  where $\mathbf{v}'$ represent the modified vertices and the operator $\|.\|$ computes the Euclidean distance;
- Peak Signal-to-Noise Ratio (PSNR): measures the distortion with respect to the maximum elongation of the vertices from the centroid of the model,

$$\mathrm{PSNR} = 10 \log_{10} \frac{\left(\max\limits_{\mathbf{v}\in \mathrm{Y}} \|\mathbf{v} - \mathbf{c}\|\right)^2}{\frac{1}{\mathrm{card}(\mathrm{Y})} \sum_{\mathbf{v}\in \mathrm{Y}} \|\mathbf{v} - \mathbf{v}'\|^2}, \tag{10}$$

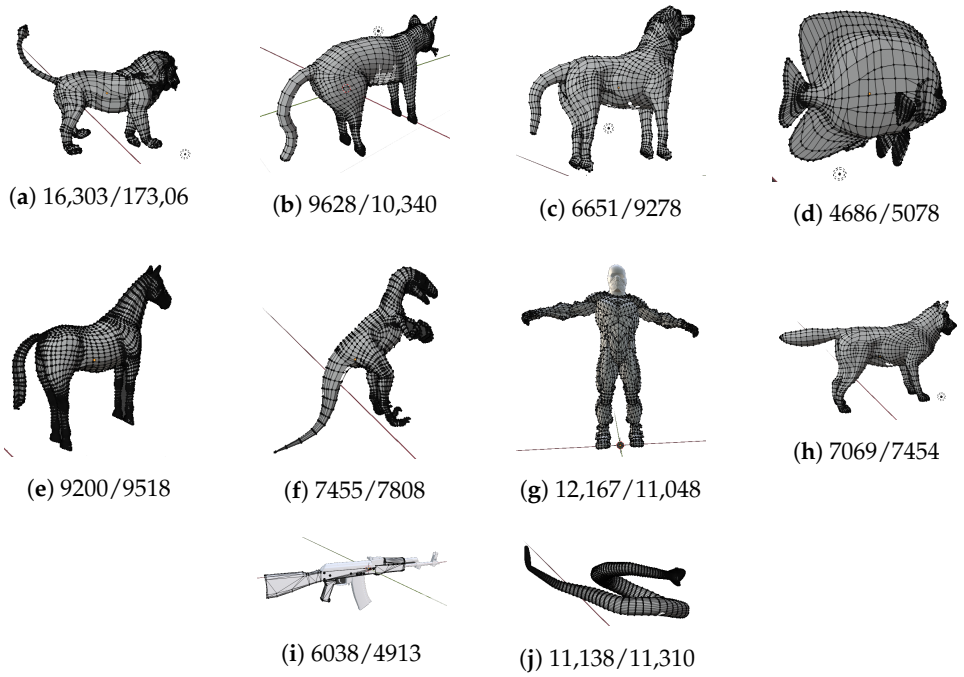
  where $\mathbf{v}'$ represent the modified vertices, $\mathbf{c}$ is the centroid of the host model, and the operator $\|.\|$ computes the Euclidean distance.

We chose a setting for the genetic algorithm to be used in all experiments; in particular, we derived the setting values from many experiments devised to have fast convergence to a solution resulting in a high quality watermarked model. The settings were population size = 300, $p_c = 0.9$, $p_m = 0.25$, maximum generations = 2000.

We ran the embedding algorithm on ten 3D models (shown in Figure 3), and the objective results are reported in Table 3. The payload in all experiments was set to 5 bpv.

From Table 3, it can be seen that, given the very high values of sensitivity for small modifications (more than 99% of vertices detected when altering them by only ± 2 in one of the bytes of their coordinates), the proposed algorithm is highly reliable in tampering detection, i.e., it is very fragile. Moreover, the very small values of MAE and RMSE and the very high values of PSNR proved that the 3D models underwent a negligible modification, which may be generally accepted apart from specific applications, in which case we think that a reversible algorithm should be applied (but this is not the objective of the present proposal).

As concerns the complexity of the proposed method, let us consider separately the preprocessing phase, the embedding phase, and the verification phase. During preprocessing, for every vertex, a signature of the polygons to which it belongs is computed. In the worst case scenario, a vertex is connected to all other vertices, so this computation would take $O(n^2)$ time, where n is the number of vertices. The embedding phase, instead, is linear in the number of vertices of the model, and it is upper bounded because the GA is run for a given predefined number of generations. Finally, the verification phase is very fast, and again, it is linear in the number of vertices (it usually takes a few hundreds of milliseconds on a standard laptop).



(**a**) 16,303/173,06 (**b**) 9628/10,340 (**c**) 6651/9278 (**d**) 4686/5078

(**e**) 9200/9518 (**f**) 7455/7808 (**g**) 12,167/11,048 (**h**) 7069/7454

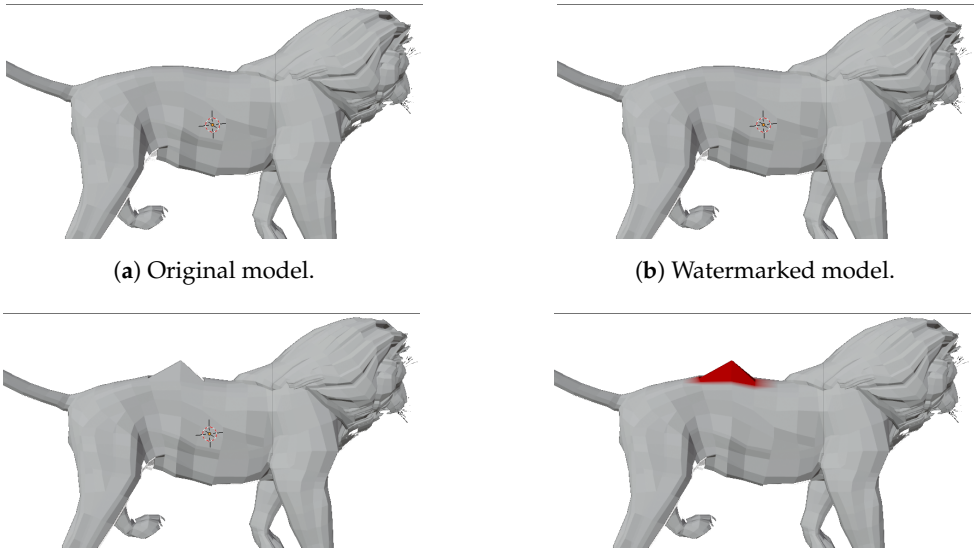
(**i**) 6038/4913 (**j**) 11,138/11,310

Figure 3. Original models used in the experiments along with [number of vertices]/[number of polygons]composing them: (**a**) lion, (**b**) cat, (**c**) dog, (**d**) fish, (**e**) horse, (**f**) raptor, (**g**) suit, (**h**) wolf, (**i**) AK, and (**j**) snake.

Table 3. Performance of the proposed algorithm on a set of ten publicly available 3D models.

3D Model	Sensitivity ± 1 (%)	Sensitivity ± 2 (%)	MAE $\times 10^{-6}$	RMSE $\times 10^{-6}$	PSNR (dB)
lion	93.85	99.82	0.65	2.76	312.85
cat	93.68	99.84	0.63	2.74	310.07
dog	93.83	99.81	0.75	3.08	310.85
fish	93.71	99.85	0.50	2.30	311.03
horse	93.79	99.84	2.30	2.96	306.21
raptor	93.82	99.83	0.69	2.95	314.29
suit	93.71	99.84	0.071	0.36	299.61
wolf	94.01	99.88	0.63	2.80	309.04
AK	93.94	99.83	1.52	8.85	317.46
snake	93.81	99.85	0.73	3.10	302.39
Average	93.82 ± 0.20	99.84 ± 0.02	0.43 ± 0.35	3.19 ± 2.04	309.36 ± 5.12

One of the original models (lion) is shown in Figure 4a. To present how the algorithm operates on the model from a subjective point of view, in Figure 4b, the same model after the fragile watermark embedding is reported.



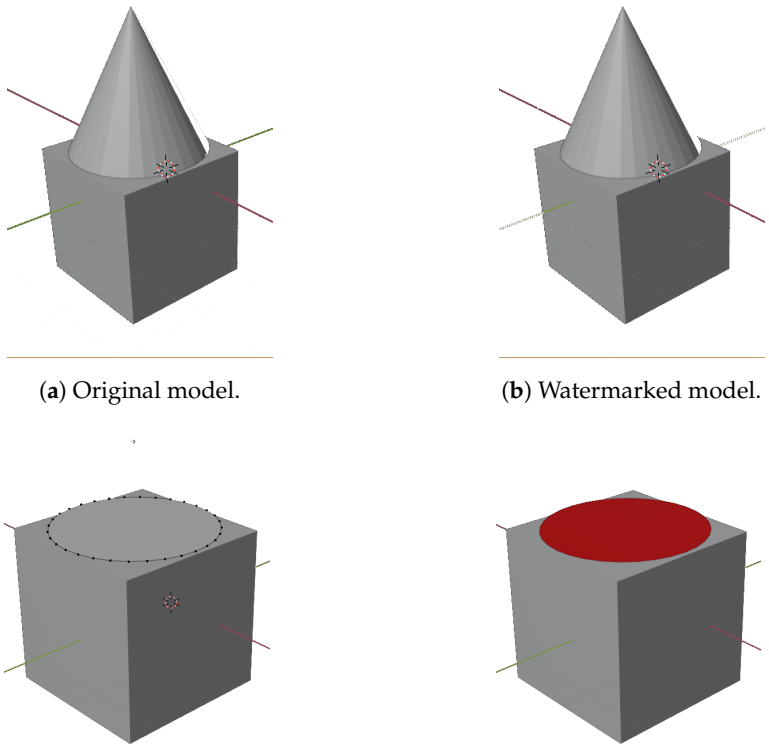
(**a**) Original model. (**b**) Watermarked model.

(**c**) Tampered model. (**d**) Detection of tampered area.

Figure 4. Application of the proposed algorithm to a model (lion).

An example of tampering is shown in Figure 4c, and the resulting tamper detection highlights the modified vertices in red color, as can be seen in Figure 4d.

To give more insights into how the algorithm detects tampering, we report a simple model in Figure 5a and show how the removal of a single vertex (Figure 5c) from the watermarked model (Figure 5b) results in a detection (Figure 5d) that highlights all the vertices contiguous to the removed one; these vertices are marked as forged because, as said in Item 2. of Section 5.3, the polygons containing them were altered: indeed, they were removed, deleting the vertex in the cone apex.



(**a**) Original model. (**b**) Watermarked model.

(**c**) Tampered model. (**d**) Detection of tampering.

Figure 5. Application of the proposed algorithm to a simplified model.

Comparison with Related Works

Among the related works, we selected for comparison the ones that reported numerical (objective) measures of distortion introduced by their algorithms, as there were no visible modifications to the model after watermark embedding. It should be pointed out that an exact comparison was not really possible as every paper used different sets of models with respect to one another, so we only compared with the best results they reported. Moreover, we emphasize that there were not many works on fragile watermarking reporting parameters to compare to in a fair way.

All in all, we can say that our proposed method was superior to all the compared methods in terms of quality (in many cases, more than two orders of magnitude of the considered measure) of the resulting models and the fragility of the watermark.

In [14], the authors reported the best PSNR value of 84.47 dB, which was far less than the smallest PSNR (>299 dB) our algorithm produced.

In [12], the authors reported distortion (the same as our MAE values) for various settings of their method's parameters; again, an exact comparison was difficult for two main reasons. First, the set of models they used was different from our set, and the model precision was 10^{-6}, whereas the precision of our models was 10^{-4}. Anyway, their best result was 1.3×10^{-5}, which was worse than our worst result (horse, 2.30×10^{-6}) by one order of magnitude. In [18], the same authors reported a distortion of 0.01, four orders of magnitude worse than our distortion results.

In [10], the authors reported a mean distance error (averaged over five models) of 2.33×10^{-4}, which was two orders of magnitude greater than the mean distance error (averaged over 10 models) computed by our models (1.92×10^{-6}).

In [15], the authors reported an average RMS error (3.74×10^{-4}) that was two orders of magnitude greater than our reported RMSE (3.19×10^{-6}), while our *RMSRatio* (1.07×10^{-7}) was comparable with theirs. It should be pointed out, anyway, that our payload (five bits per vertex) was five times that of [15], but the distortion induced was the same. Moreover, the work in [15] had the disadvantage of requiring the centroid of the original model to perform the integrity verification, thus requiring out-of-band information to be kept per-model.

In [20], the authors reported an average (over five models) SNR of 124.24 dB that was far smaller than our average (over 10 models) SNR of 231.14 dB.

Finally, we performed several other experiments on the same models with different precisions, and in general, we could say that the proposed method resulted in a distortion that was in the order of $1/100^{\text{th}}$ of the precision of the original model.

7. Discussion

The proposed fragile watermarking algorithm for 3D models performed a watermark bit string embedding by modifying the least significant part of some vertices' attributes (in the present embodiment, coordinates): the watermark was stored in a secret space defined by a secret KLT basis and protected the integrity of the structure of the 3D model, that is the polygon mesh and the vertices of these polygons. Thus, the security of the method lied in the secrecy of the vectors used to derive the KLT kernel.

The main advantages of the method were:

- the proposed method was very general because it worked on single vertices and may be extended to protect any number of vertex attributes (like normals, texture properties, etc.), even without modifying the KLT basis size: for example, by applying the cryptographic hash function also on these attributes and XORing the result to the fingerprint F;
- the watermark was embedded into the vertices without tying to a particular representation of the model, i.e., vertices could be changed in order as long as the polygons were updated accordingly;
- modifications to the data were limited to the least significant bits of the mantissa value, so keeping the relative error as small as possible;

- affine transformations to the 3D models were allowed (no tampering was shown), as long as they were represented and stored as matrix operations to be applied to vertex coordinates (this is the default Blender behavior).

At the same time, the disadvantages of the method were:

- a slight change to the vertices' coordinates was performed;
- if a vertex was duplicated in the 3D model, after watermark embedding, two vertices with different coordinates and/or normal components would be created; thus, small holes or changes in illumination in the vertex area were created; this issue could be solved by recognizing coincident vertices and modifying them in the same manner during watermark embedding; it was possible to perform this action because it was highly probable that both vertices allowed for an equal modification embedding different watermark strings.

7.1. Security Considerations

The security of the method was based on the secrecy of the KLT basis. Thus, this set of orthonormal vectors constituted the secret key. This basis may be derived from any sample of vectors belonging to a vector space of the right dimension: in our case, the EU was composed of 28 bytes; thus, the vector space should be of dimension 28.

For example, considering a grayscale image and building vectors from pixel values taken in raster scan order allowed the computation of a covariance matrix from which to derive an orthonormal basis (as shown in Section 3): if the image were kept secret, so would be the KLT basis and consequently the embedding space of the watermark bit string. This prevented an attacker from creating forged vertices embedding a correct watermark given that she/he would not know the kernel basis of the transform.

If the length of the watermark stored in each EU were m bits (i.e., the payload was m bpv), then the probability that a forged vertex would not be detected as such would be $1/2^m$. Nonetheless, as previously said, if the modification were in the higher part of the vertex data, then all the polygons sharing that vertex would detect the modification: given that for each polygon, the probability to fail detection would be $1/2^m$, if the forged vertex were shared amongst k polygons, then the probability that the tampering went undetected by all polygons would be $1/2^{mk}$.

8. Conclusions

The proposed method performed an integrity protection of 3D models in a more secure embedding space of the watermark while reducing the distortion induced to the model. It applied a small modification to some attributes of the vertices of the model (in the current implementation, vertices' coordinates, but it could be extended to other vertex features). The error introduced w.r.t. the original attribute values was in the order of 4.3×10^{-7} (average MAE for the models analyzed in Table 3) and was not noticeable from the rendered model (as subjectively judged from a sample of 10 observers).

The security of the whole process lied in the secrecy of the KLT basis, which, in turn, was based on the secrecy of the vectors used to compute it. Public parameters of the algorithm were the cryptographic hash function, the payload m (in bpv), the orders of the m KLT coefficients, and the embedding position p.

As future work, we plan to extend the algorithm to protect other vertex attributes, like uv coordinates and normals.

Author Contributions: Conceptualization: M.B., D.C. and M.G.; Methodology: M.B. and D.C.; Software: M.B.; Data Curation: P.P.;writing—original draft preparation, M.B., D.C., M.G. and P.P.; writing—review, M.B., D.C., M.G. and P.P. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Cox, I.; Miller, M.; Bloom, J.; Fridrich, J.; Kalker, T. *Digital Watermarking and Steganography*, 2nd ed.; Morgan Kaufmann: San Francisco, CA, USA, 2007.
2. Praun, E.; Hoppe, H.; Finkelstein, A. Robust mesh watermarking. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '99*; ACM Press/Addison-Wesley Publishing Co.: New York, NY, USA, 1999; pp. 49–56. [CrossRef]
3. Ohbuchi, R.; Takahashi, S.; Miyazawa, T.; Mukaiyama, A. Watermarking 3D Polygonal Meshes in the Mesh Spectral Domain. In Proceedings of the Graphics Interface 2001 Conference, Ottawa, ON, Canada, 7–9 June 2001; pp. 9–18.
4. Ohbuchi, R.; Mukaiyama, A.; Takahashi, S. A Frequency-Domain Approach to Watermarking 3D Shapes. *Comput. Graph. Forum* **2002**, *21*, 373–382. [CrossRef]
5. Barni, M.; Bartolini, F.; Cappellini, V.; Corsini, M.; Garzelli, A. Digital watermarking of 3D meshes. Mathematics of Data/Image Coding, Compression, and Encryption VI, with Applications. *Int. Soc. Opt. Photonics* **2004**, *5208*, 68–79.
6. Yu, Z.; Ip, H.H.; Kwok, L. A robust watermarking scheme for 3D triangular mesh models. *Pattern Recognit.* **2003**, *36*, 2603–2614. [CrossRef]
7. Li, L.; Zhang, D.; Pan, Z.; Shi, J.; Zhou, K.; Ye, K. Watermarking 3D mesh by spherical parameterization. *Comput. Graph.* **2004**, *28*, 981–989. [CrossRef]
8. Cho, J.W.; Prost, R.; Jung, H.Y. An Oblivious Watermarking for 3-D Polygonal Meshes Using Distribution of Vertex Norms. *IEEE Trans. Signal Process.* **2007**, *55*, 142–155. [CrossRef]
9. Vasic, B.; Raveendran, N.; Vasic, B. Neuro-OSVETA: A Robust Watermarking of 3D Meshes. In *International Telemetering Conference Proceedings*; International Foundation for Telemetering: San Diego, CA, USA, 2019; Volume 55.
10. Wang, Y.P.; Hu, S.M. A New Watermarking Method for 3D Models Based on Integral Invariants. *IEEE Trans. Vis. Comput. Graph.* **2009**, *15*, 285–294. [CrossRef]
11. Yeung, M.; Yeo, B.L. Fragile watermarking of Three-Dimensional Objects. In Proceedings of the International Conference on Image Processing, ICIP98, Chicago, IL, USA, 7 October 1998; Volume 2, pp. 442–446.
12. Chou, C.M.; Tseng, D.C. A Public Fragile Watermarking Scheme for 3D Model Authentication. *Comput.-Aided Des.* **2006**, *38*, 1154–1165. [CrossRef]
13. Wang, W.B.; Zheng, G.Q.; Yong, J.H.; Gu, H.J. A Numerically Stable Fragile Watermarking Scheme for Authenticating 3D Models. *Comput.-Aided Des.* **2008**, *40*, 634–645. [CrossRef]
14. Su, Z.; Li, W.; Kong, J.; Dai, Y.; Tang, W. Watermarking 3D CAPD models for topology verification. *Comput.-Aided Des.* **2013**, *45*, 1042–1052. [CrossRef]
15. Huang, C.C.; Yang, Y.W.; Fan, C.M.; Wang, J.T. A Spherical Coordinate Based Fragile Watermarking Scheme for 3D Models. In *International Conference on Industrial, Eengineering and other Applications of Applied Intelligent Systems*; Springer: Berlin/Heidelberg, Germany, 2013; pp. 566–571.
16. Yeo, B.L.; Yeung, M.M. Watermarking 3D objects for verification. *IEEE Comput. Graph. Appl.* **1999**, *19*, 36–45.
17. Lin, H.Y.S.; Liao, H.Y.M.; Lu, C.S.; Lin, J.C. Fragile watermarking for authenticating 3-D polygonal meshes. *IEEE Trans. Multimed.* **2005**, *7*, 997–1006. [CrossRef]
18. Chou, C.M.; Tseng, D.C. Affine-Transformation-Invariant Public Fragile Watermarking for 3D Model Authentication. *IEEE Comput. Graph. Appl.* **2009**, *29*, 72–79. [CrossRef] [PubMed]
19. Wu, H.T.; Cheung, Y.M. A Fragile Watermarking Scheme for 3D Meshes. In Proceedings of the 7th Workshop on Multimedia and Security, New York, NY, USA, 1–2 August 2005; ACM: New York, NY, USA, 2005; pp. 117–124. [CrossRef]
20. Motwani, M.; Motwani, R.; Frederick Harris, J. Fragile Watermarking of 3D Models Using Genetic Algorithms. *J. Electron. Sci. Technol.* **2010**, *8*, 244–250.
21. Gonzalez, R.C.; Wintz, P. *Digital Image Processing*, 2nd ed.; Addison-Wesley Publishing Co., Inc.: Reading, MA, USA, 1987.
22. Goldberg, D.E. *Genetic Algorithms in Search, Optimization and Machine Learning*; Addison-Wesley Longman Publishing Co., Inc.: Boston, MA, USA, 1989.

23. Hassanien, A.E.; Abraham, A.; Kacprzyk, J.; Peters, J.F. Computational intelligence in multimedia processing: Foundation and trends. In *Computational Intelligence in Multimedia Processing: Recent Advances*; Springer: Berlin/Heidelberg, Germany, 2008; pp. 3–49.
24. Botta, M.; Cavagnino, D.; Pomponiu, V. A modular framework for color image watermarking. *Signal Process.* **2016**, *119*, 102–114. [CrossRef]